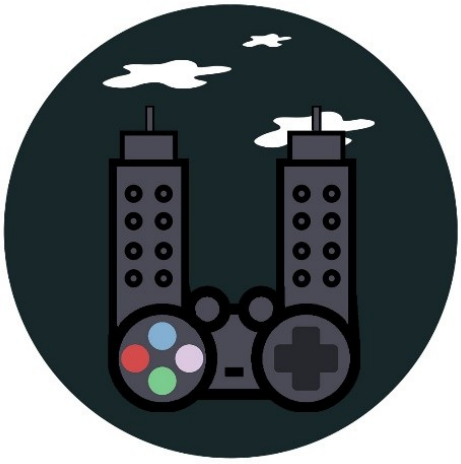


GAMELAND

Progetto F.I.A.

Presentato da	Savino Paolo, Di Genio Gaetano, Iaccarino Emanuele, Balbi Vito

SOMMARIO:

1. Sistema attuale e sistema proposto

Il nostro progetto nasce per dare la possibilità agli appassionati di videogames (e non) di poter acquistare i migliori videogiochi disponibili comodamente da casa, con un'esperienza quanto più semplice e piacevole possibile, in copia digitale o fisica, mediante pochi e facili click.
Il nostro team si impegna a realizzare un progetto che sia facile ed intuitivo al cliente, in modo tale da poter assistere, semplificare e velocizzare l'acquisto di un videogioco.
Esistono difatti molteplici piattaforme per la vendita di prodotti inerenti al mondo dei videogiochi, alcune delle quali estremamente popolari (G2A, GameStop, InstantGaming, …). Tuttavia, nonostante la loro popolarità, la maggior parte di queste piattaforme non dispone di un sistema che riesca a rendere la loro esperienza di utilizzo da parte degli utenti realmente unica.
Le attuali piattaforme che si occupano della vendita di videogiochi permettono all'utente di ricercare solo manualmente il prodotto videoludico a cui è interessato, proprio per questo problema il nostro progetto dà la possibilità di poter acquistare videogiochi a chiunque in modo semplice ed efficiente. Tutto questo è possibile attraverso il supporto di un sistema che agevoli quelle che sono le richieste degli utenti e renda tale esperienza facile da portare a termine.
Questo supporto viene dato tramite l'utilizzo di un modulo di Intelligenza Artificiale che sia in grado di consigliare i videogiochi inerenti a quelli che l'utente sta visualizzando in un preciso momento.

---

2. Descrizione dell'agente

2.1 Obiettivi

Lo scopo del nostro progetto è quello di realizzare un modulo di intelligenza artificiale, e quindi un agente intelligente, in grado di consigliare prodotti, nello specifico prodotti videoludici, affini a quello che l'utente sta visualizzando.

2.2 Specifica PEAS

PEAS	
Performance	Capacità di consigliare prodotti quanto più affini a quello che l'utente sta visualizzando e quindi consigliare prodotti interessanti per quell'utente.
Environment	• **Dinamico**, in quanto i prodotti consigliati cambiano in base a ciò che l'utente sta visualizzando; • **Sequenziale**, in quanto le azioni passate degli utenti influenzano le decisioni future dell'agente; • **Discreto**, perché il prezzo di un prodotto assume un valore approssimato a due cifre dopo la virgola; • **Completamente osservabile**, dato che si ha accesso a tutte le informazioni relative ai prodotti; • **Deterministico**, in quanto lo stato successivo è determinato dallo stato attuale e dall'azione eseguita dall'agente; • A **singolo agente**, in quanto l'unico agente che opera in questo ambiente è quello in oggetto.
Actuators	Gli attuatori dell'agente consistono nei 2 diversi prodotti consigliati sulla base del prodotto che l'utente sta visualizzando.
Sensors	I sensori dell'agente consistono nella scheda di un determinato prodotto.

2.3 Analisi del problema

Abbiamo deciso di affrontare il problema consigliando all'utente due prodotti sulla base di quello che sta visualizzando, quindi inerente ai suoi interessi.

---

3. Raccolta, analisi e preprocessing dei dati

3.1 Scelta del dataset

A questo punto, è toccato scegliere il dataset per poter effettivamente creare il modello di machine learning.

Per fare ciò, abbiamo cercato in rete dataset già realizzati che meglio si adattassero alle nostre esigenze, ricercandoli in siti rinomati quali kaggle.com, AWS, Google Dataset Search Engine, Microsoft Datasets ed altri siti minori.

La maggior parte dei dataset trovati sono stati scartati poiché:

- Troppi pochi dati, nell'ordine delle poche centinaia.
- Dataset troppo grandi, nell'ordine di Tb di memoria e centinaia di migliaia di elementi.
- Dataset con quantità di dati non eccessiva e non troppo scarna, ma nel complesso non adatti ai nostri scopi.

Come seconda opzione si sarebbe potuto realizzare un dataset interamente da 0, in modo tale da avere il totale controllo sui dati da inserire e poterlo adattare al meglio alle nostre esigenze.

Tuttavia, sarebbe risultato estremamente faticoso e costoso in termini di tempo, per cui abbiamo preferito abbandonare questa strategia e concentrarci interamente sulla prima, sicuramente più conveniente dato il tempo e le risorse a nostra disposizione.

3.2 Scrematura ed analisi del dataset

Il dataset scelto proviene dal sito kaggle.com ed era inizialmente composto da circa 10mila elementi.

Il nostro progetto prevede la creazione di un modulo di intelligenza artificiale in grado di consigliare agli utenti prodotti inerenti ai suoi interessi, nello specifico, inerenti al prodotto della pagina che sta visualizzando.

Siccome il dataset da noi scelto era composto anche da un gran numero di videogiochi realizzati per console datate quali Gamecube, Ps1, Nintendo 64 ecc. abbiamo dovuto, nostro malgrado, "tagliarli fuori" per coerenza con le motivazioni sopra esposte.

Ne consegue un dataset di dimensioni assai più ridotte dell'originale, ma comunque sufficiente per poter raggiungere i nostri scopi: circa 3000 prodotti disponibili in versione digitale e fisica.

Il dataset è così composto dalle seguenti colonne:

Colonna Name:

Name

Contiene essenzialmente il nome del prodotto (videogioco).

Colonna Platform:

Platform

Contiene la piattaforma (console) per la quale è realizzato il videogioco. Come detto in precedenza, abbiamo effettuato operazioni di taglio per quelle console eccessivamente datate, per la quale non è prevista la doppia tipologia di videogioco (digitale / fisico).

Colonna Year_of_Release:

Year_of_Release

Contiene l'anno di rilascio del videogioco per la relativa console.

Colonna Genre:

Genre

Contiene il genere del videogioco relativo, ad esempio: Sport, Racing, Simulation, Shooter, Role-Playing, Action, Platform, Misc, Fighting.

Colonna Publisher:

Publisher

Contiene l'informazione riguardante la casa produttrice del videogioco, ad esempio: Nintendo, Activision, Sony, Ubisoft, Take-Two Interactive, Electronic Arts, Sega.

Colonna Price:

Price

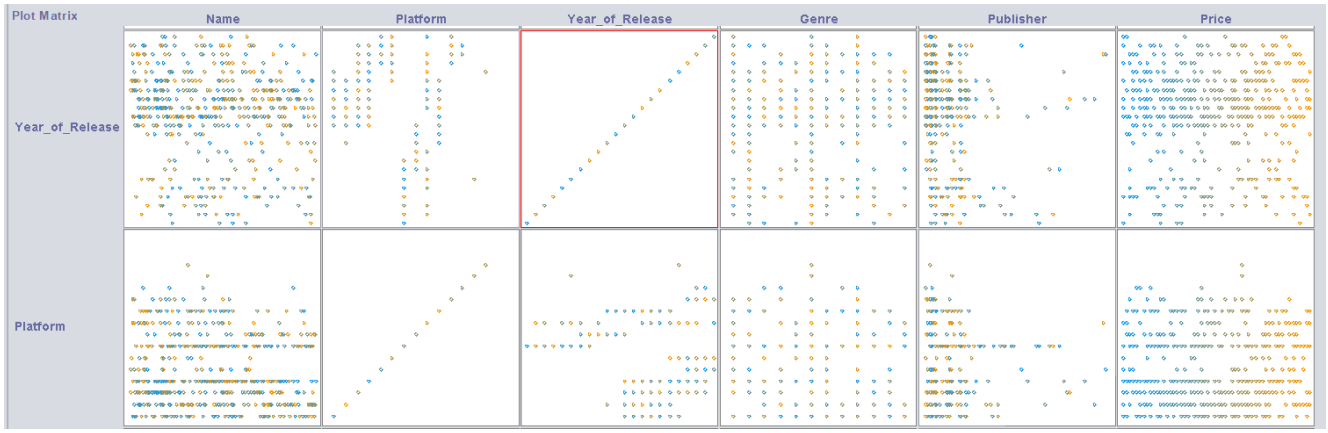
Contiene l'informazione riguardante il prezzo del videogioco, in forma intera senza decimi.

4. Algoritmo di clustering

Arrivati a questo punto, ciò che dovevamo fare era scegliere l'algoritmo di clustering da utilizzare e creare la parte di codice relativa ad esso.

Per testare i due algoritmi che abbiamo scelto, DBSCAN e K-MEANS, abbiamo utilizzato, in aggiunta alle parti di codice, l'interfaccia grafica di weka dove abbiamo riscontrato subito un problema:il nostro dataset era molto confusionario, poco preciso e poco accurato. Purtroppo, tra tutti i dataset trovati, era quello più adatto alle nostre esigenze e crearlo da zero sarebbe stato troppo impegnativo ed inutile per il nostro scopo.

A prova di tutto ciò che viene detto rendiamo visibile la disposizione dei nostri dati:



4.1 DBSCAN

Il primo algoritmo che abbiamo testato è stato il **DBSCAN**.

Il **DBSCAN** è un algoritmo che si basa sul concetto di densità dei campioni. Utilizza due parametri molto importanti che sono minPts (ovvero il numero di punti per considerare denso l'interno di un punto) e **ϵ** (ovvero la distanza che definisce un intorno circolare di ciascun punto dai suoi vicini.

A causa della mancanza di etichette però per misurare il potenziale del raggruppamento si può ricorrere a 3 modalità:

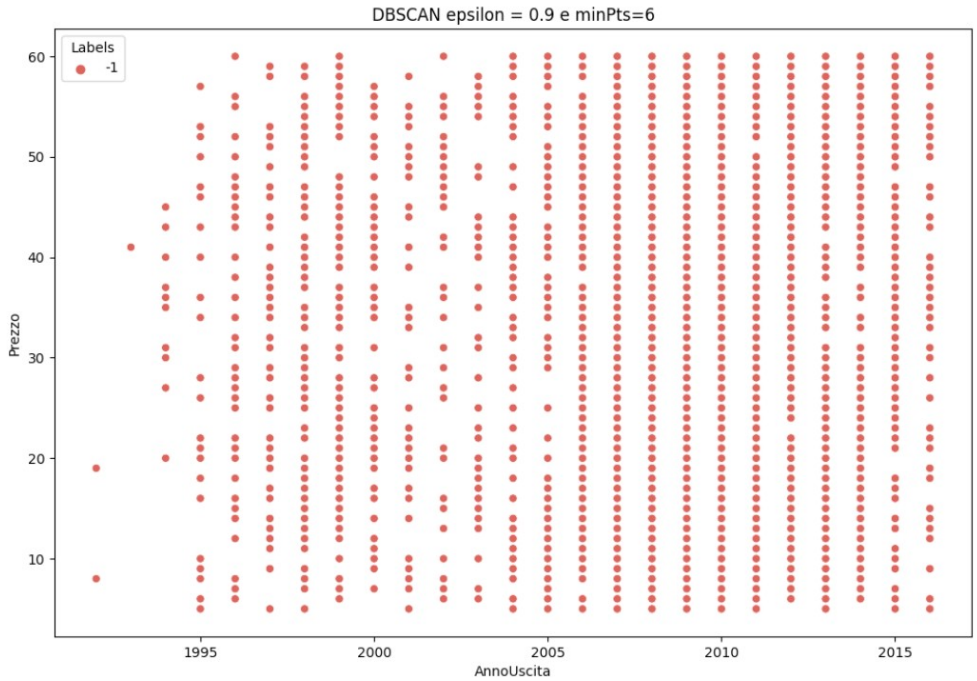
- Punto a gomito
- Silhouette score
- Adjusted Rand Index

Non siamo riusciti a ricrearlo in java ma lo abbiamo testato creato attraverso un semplice script in Python e tramite l'interfaccia grafica di WEKA

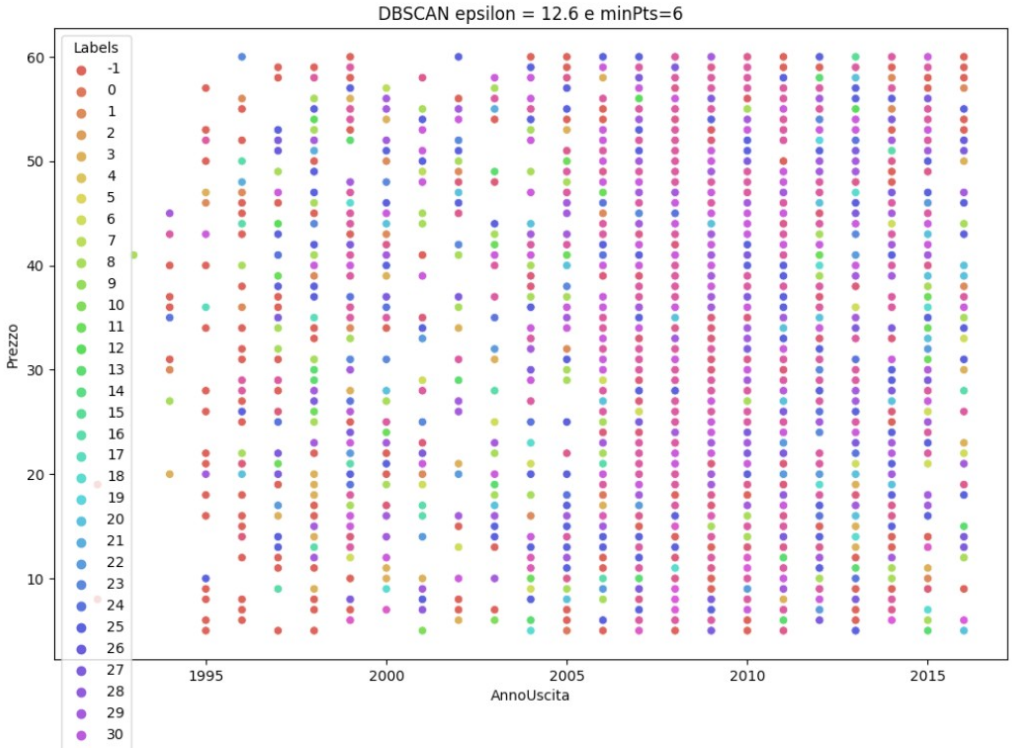
Script python:

```python
import pandas as pd
from sklearn.cluster import DBSCAN
import json
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('../../Desktop/FIA/datasetFinito.csv')
X = df.drop(['Titolo'], axis=1)

db = DBSCAN(eps=12.6, min_samples=6).fit(X)

X['Labels'] = db.labels_
plt.figure(figsize=(12, 8))
sns.scatterplot(X['AnnoUscita'], X['Prezzo'], hue=X['Labels'], palette=sns.color_palette('hls', np.unique(db.labels_).shape[0]))
plt.title('DBSCAN epsilon = 12.6 e minPts=6')
plt.show()
```

DBSCAN con epsilon = 0.9 e minPts=6 (valori di default)

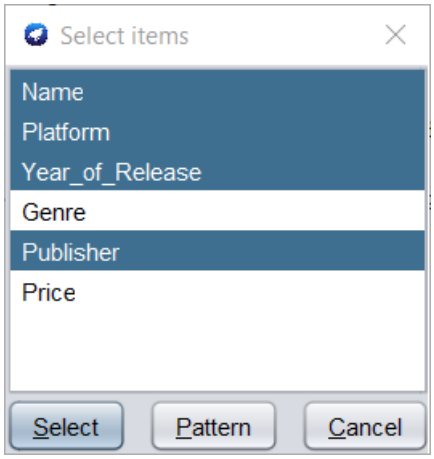


DBSCAN con epsilon = 12,6 e minPts=6

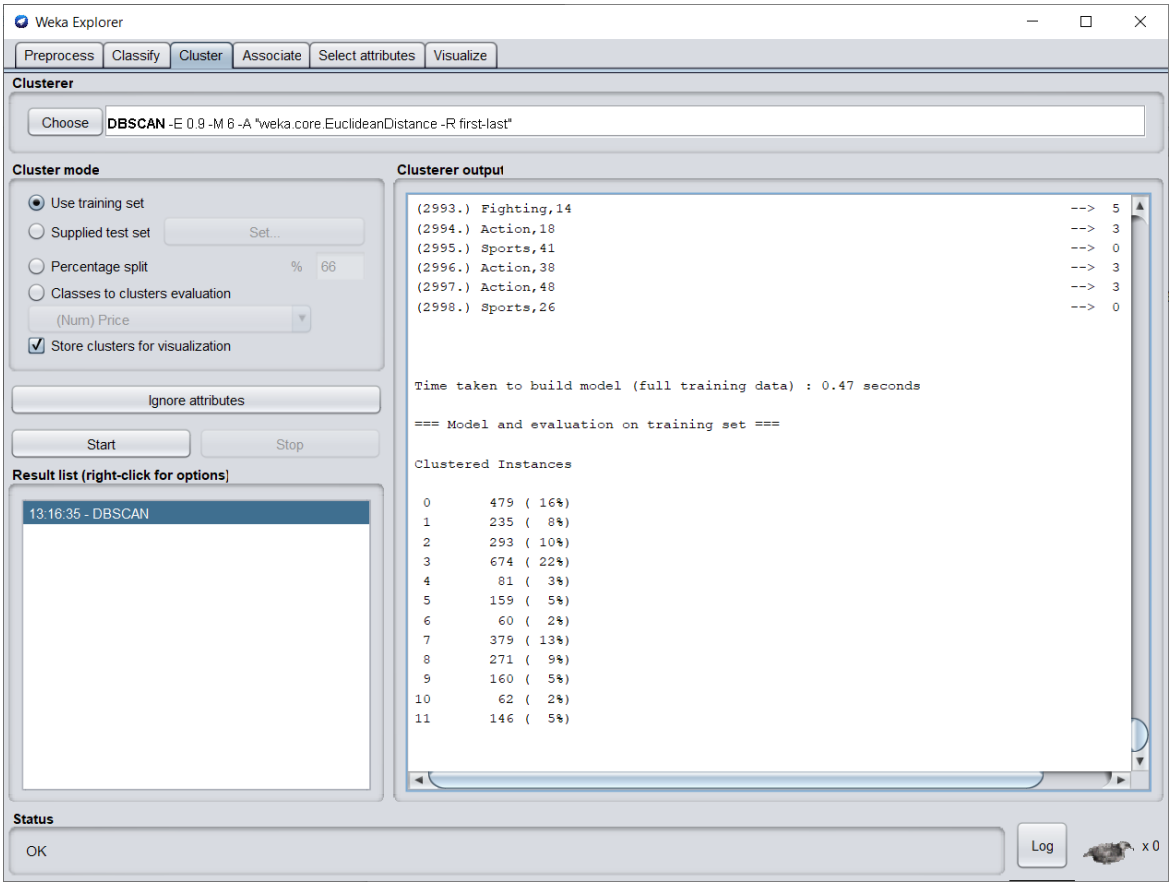


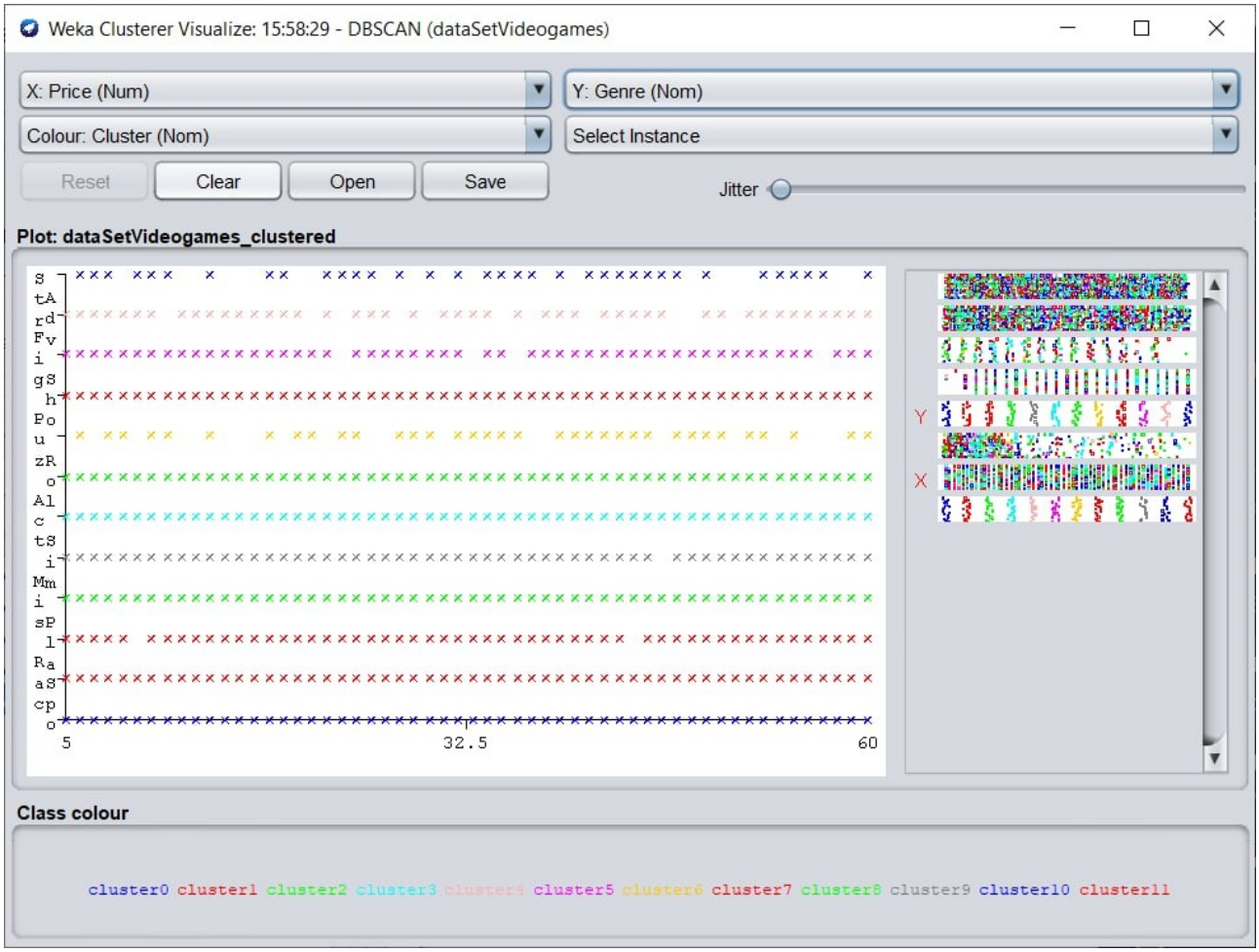
All’interno di weka abbiamo preferito lasciare l’epsilon ed il minPts di default (0,9 e 6) ed abbiamo ottenuto degli esiti totalmente differenti rispetto a ciò che abbiamo ottenuto con lo script python pur inserendo gli stessi valori

All’interno di Weka, per effettuare l’algoritmo di clustering DBSCAN abbiamo deciso di ignorare alcuni attributi

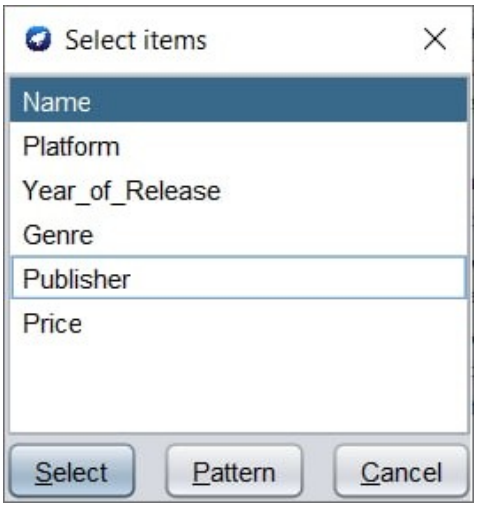


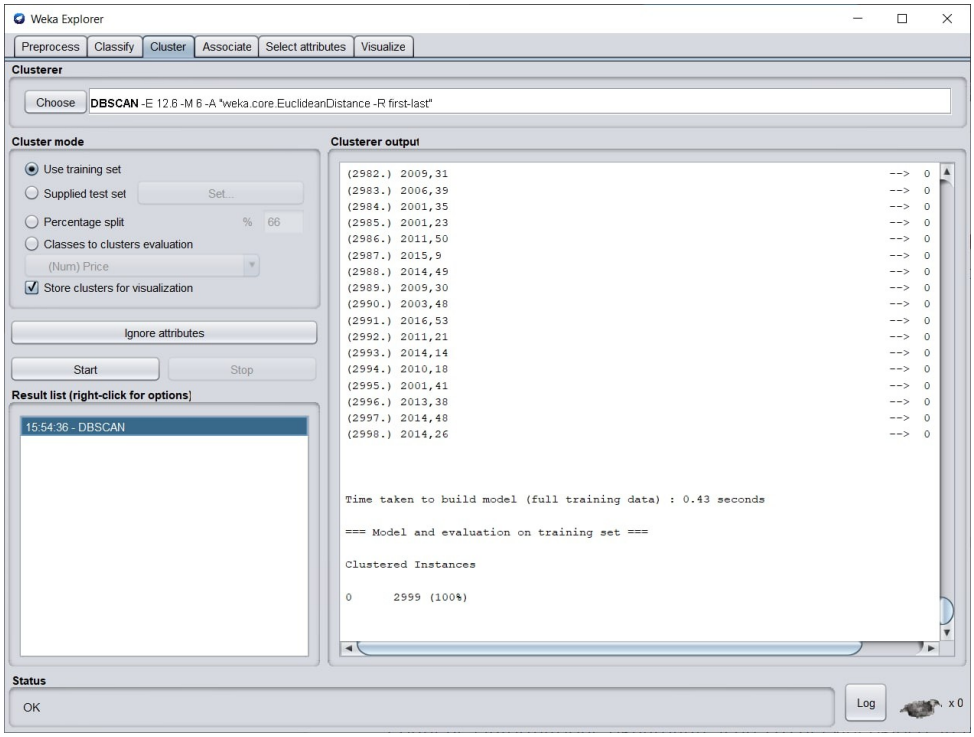
Ed i risultati che abbiamo ottenuto sono stati questi:

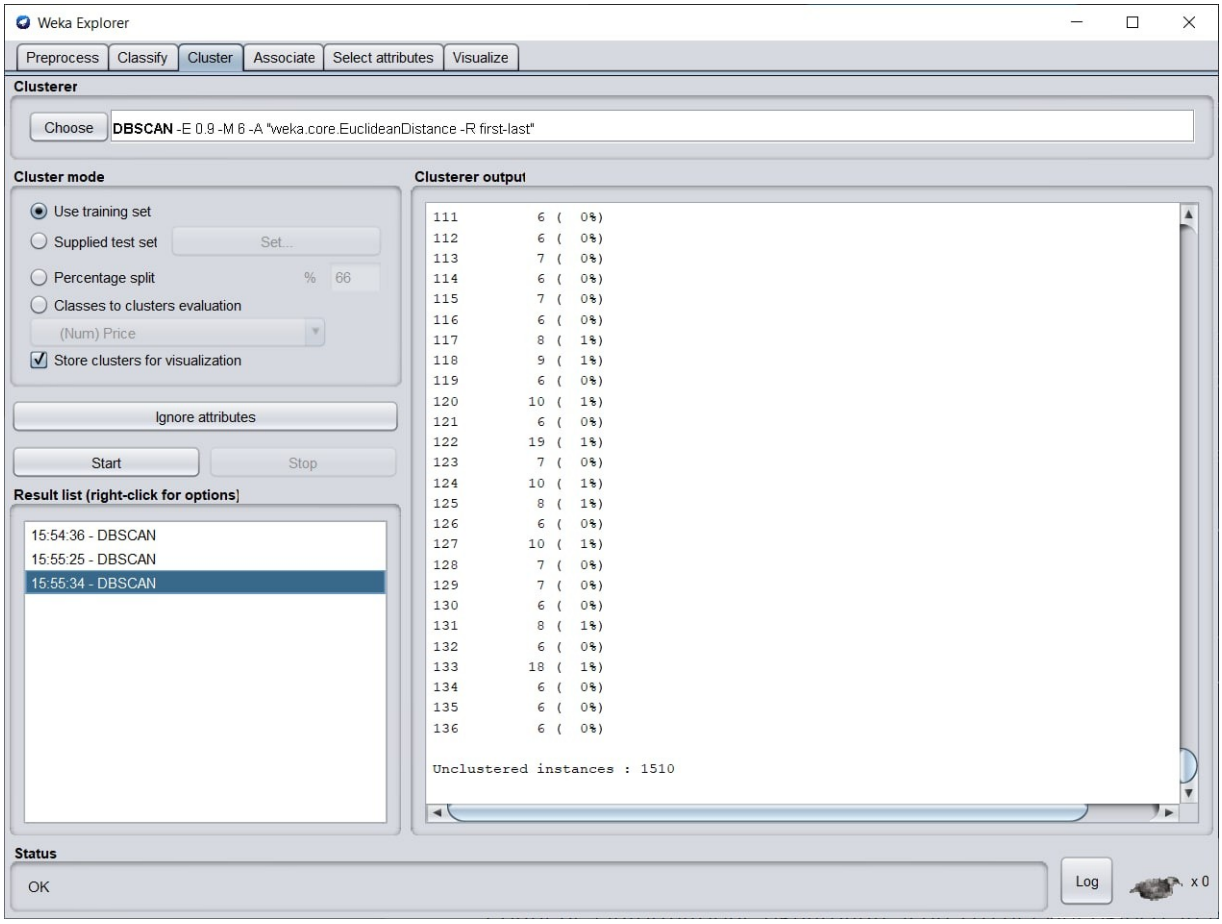



Weka Clusterer Visualize: 15:58:29 - DBSCAN (dataSetVideogames)
X: Price (Num)
Y: Genre (Nom)
Colour: Cluster (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: dataSetVideogames_clustered
5
32.5
60
Y
X
Class colour
cluster0 cluster1 cluster2 cluster3 cluster4 cluster5 cluster6 cluster7 cluster8 cluster9 cluster10 cluster11

Provando altre combinazioni (es. ignorando solo il nome) con valori epsilon e minPts di default (0,9 e 6) si ottengono 136 cluster mentre con valori epsilon e minPts di (12,6 e 6) si ottengono 1 cluster. Entrambi i risultati non sono per noi ammissibili:







Valori analoghi sono stati ottenuti impostando altre combinazioni, per cui ne ricaviamo una totale inadeguatezza del dataset per questo algoritmo.

Abbiamo deciso di scartare il **DBSCAN** per altri due motivi:

1. Difficoltà nell'implementazione.

2. Non possiede nessuna rappresentazione dei cluster, che risultava importante per risalire ai prodotti da consigliare agli utenti (era necessario conoscere le caratteristiche più comuni dei prodotti).

4.2 K-MEANS

Queste ragioni ci hanno portato a scegliere l'algoritmo di clustering **K-MEANS**.

Il problema di questo algoritmo è proprio dover individuare a priori il numero di cluster. Abbiamo implementato uno script in java per determinare il numero di cluster, il seed (randomizzazione dei punti dei cluster iniziali) e il silhouette score. Abbiamo scelto di provare da 2 a 30 cluster e da 0 a 200 per quanti riguarda il valore del seed.

Successivamente abbiamo creato il vero e proprio algoritmo di clustering in java ed in python, valutando il silhouette score, ed utilizzando l'interfaccia grafica di WEKA.

Algoritmo per determinare k (numero dei cluster), seed e silhouette in java:

```
int bestK = 0, bestSeed = 0;
double max = 0;

for(int i = 0; i <= 200; i++) {
    for(int j = 2; j <= 30; j++) {

        EuclideanDistance df2 = new EuclideanDistance();
        df2.setAttributeIndices("first-last");
        SimpleKMeans cl2 = new SimpleKMeans();
        cl2.setSeed(i);
        cl2.setNumClusters(j);
        cl2.setDistanceFunction(df2);
        cl2.buildClusterer(dataSet);

        SilhouetteIndex si = new SilhouetteIndex();
        si.evaluate(cl2, cl2.getClusterCentroids(), dataSet, df2);
        BigDecimal valueSiluette = (Double.isNaN(si.getGlobalSilhouette())) ? BigDecimal.valueOf(0) : BigDecimal.valueOf(si.getGlobalSilhouette());

        if ( valueSiluette.compareTo(BigDecimal.valueOf(max)) == 1) {
            max = Double.parseDouble(valueSiluette.toString());
            bestK = j;
            bestSeed = i;
        }

    }
}

System.out.println("bestK: " + bestK + " bestSeed: " + bestSeed + " maxSiluette: " + max);
```

Risultato:

```
bestK: 29 bestSeed: 82 maxSiluette: 0.07798872078816425
```

K-Means java:

```java
public class Cluster {

    public static void main(String[] args) throws Exception {

        CSVLoader loader = new CSVLoader();
        loader.setSource(new File( pathname: "src/main/webapp/dataSet/dataSetVideogames.csv"));
        Instances dataSet = loader.getDataSet();

        EuclideanDistance df = new  EuclideanDistance();
        df.setAttributeIndices("first-last");

        //KMEANS

        SimpleKMeans simpleKMeans = new SimpleKMeans();
        simpleKMeans.setSeed(82);
        simpleKMeans.setNumClusters(29);

        int[] attributes = new int[1];
        attributes[0] = 0;

        Remove remove = new Remove();
        remove.setAttributeIndicesArray(attributes);
        remove.setInputFormat(dataSet);
        dataSet = Filter.useFilter(dataSet, remove);

        simpleKMeans.buildClusterer(dataSet);
        System.out.println(simpleKMeans);

        ClusterEvaluation eval = new ClusterEvaluation();
        eval.setClusterer(simpleKMeans);
        eval.evaluateClusterer(dataSet);
        System.out.println(eval.clusterResultsToString());
```

In più, come si evince dalla foto abbiamo anche effettuato l'evalutation dei cluster, ovvero la valutazione e questo è il risultato:

```
 0        63 (  2%)
 1        64 (  2%)
 2       234 (  8%)
 3        82 (  3%)
 4       134 (  4%)
 5       163 (  5%)
 6       186 (  6%)
 7       160 (  5%)
 8        39 (  1%)
 9       142 (  5%)
10        62 (  2%)
11        88 (  3%)
12        94 (  3%)
13        80 (  3%)
14        44 (  1%)
15       176 (  6%)
16        72 (  2%)
17       160 (  5%)
18        82 (  3%)
19       150 (  5%)
20        56 (  2%)
21        78 (  3%)
22        85 (  3%)
23        87 (  3%)
24       103 (  3%)
25        57 (  2%)
26       119 (  4%)
27        88 (  3%)
28        51 (  2%)
```

Algoritmo silhouette score Python:

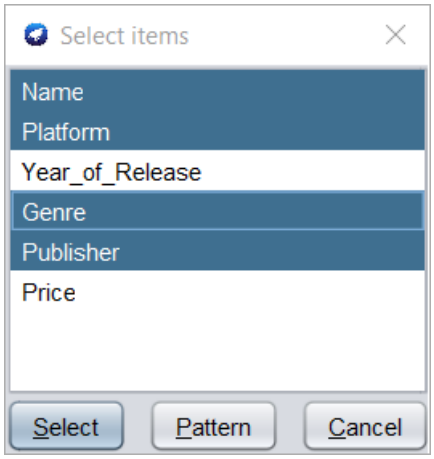
```python
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import json
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('../../Desktop/FIA/datasetFinito.csv')
X = df.drop(['Titolo'], axis=1)

clusterErrors = []

for i in range (2, 30):
    km=KMeans(n_clusters=i, max_iter=1000).fit(X)
    clusterErrors.append(km.inertia_)
    y_predict = km.fit_predict(X)
    centroids = km.cluster_centers_
    label = km.predict(X)
    print(f'Silhouette Score(n={i}): {silhouette_score(X,label)}')

fig, ax = plt.subplots(figsize=(12, 8))
sns.lineplot(x=list(range(2, 30)), y=clusterErrors, ax=ax)
ax.set_title('Punto a gomito')
ax.set_xlabel('Clusters')
ax.set_ylabel('Inertia')
```

Risultato:

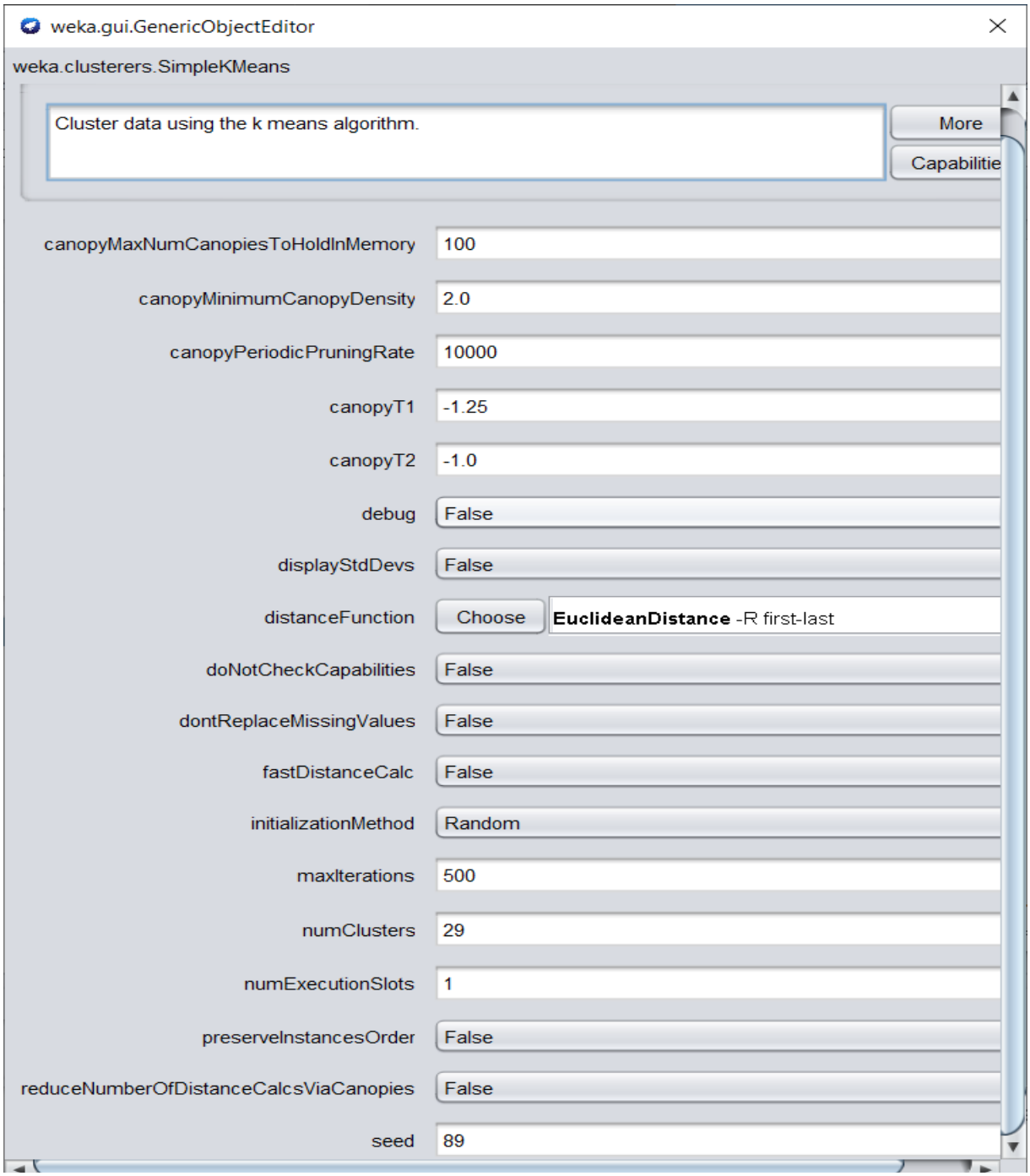
```
Silhouette Score(n=2): 0.6254868581896968
Silhouette Score(n=3): 0.5876808040798299
Silhouette Score(n=4): 0.567845250676903
Silhouette Score(n=5): 0.5551088184593373
Silhouette Score(n=6): 0.5457636216596652
Silhouette Score(n=7): 0.5383776152699137
Silhouette Score(n=8): 0.5319802652183204
Silhouette Score(n=9): 0.5264039971866126
Silhouette Score(n=10): 0.5213370835692356
Silhouette Score(n=11): 0.5167217909311175
Silhouette Score(n=12): 0.5120812869493263
Silhouette Score(n=13): 0.5076412504731397
Silhouette Score(n=14): 0.5034835842186411
Silhouette Score(n=15): 0.4998425303885978
Silhouette Score(n=16): 0.49566224205525633
Silhouette Score(n=17): 0.4913768403663523
Silhouette Score(n=18): 0.4877845091025514
Silhouette Score(n=19): 0.48347827249105874
Silhouette Score(n=20): 0.47960365740253824
Silhouette Score(n=21): 0.47544776222443824
Silhouette Score(n=22): 0.47185738876694944
Silhouette Score(n=23): 0.4681528271835041
Silhouette Score(n=24): 0.46425214054804875
Silhouette Score(n=25): 0.46058126188468707
Silhouette Score(n=26): 0.45690405975014464
Silhouette Score(n=27): 0.4526687181341023
Silhouette Score(n=28): 0.4486919271806978
Silhouette Score(n=29): 0.44586379270472015
```

K-MEANS (WEKA):

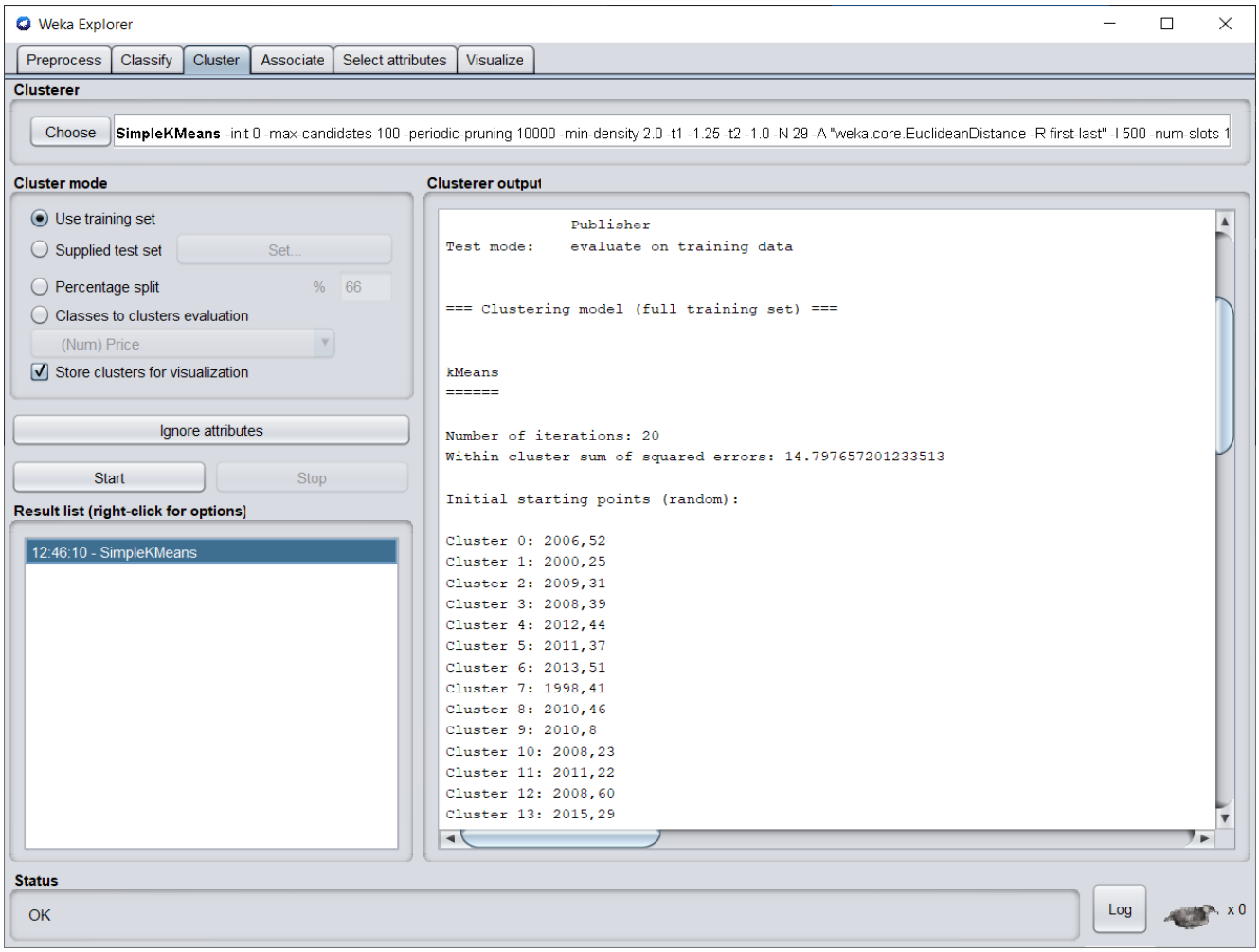
Abbiamo ignorato questi attributi



Settato questi valori:



E questo è ciò che abbiamo ottenuto:



In alternativa, abbiamo pensato di optare per algoritmi di apprendimento di tipo supervisionato, in particolare l'**SVM**.

Questa scelta, però, ci avrebbe portato ad affrontare un problema di classificazione e non più di clustering, andando del tutto fuori dagli scopi del nostro progetto.

Sulla base di queste motivazioni, abbiamo deciso di abbandonare completamente questa scelta.

5. Allenamento predizione ed integrazione del sistema

Arrivati a questo punto l'unica cosa che dovevamo fare era allenare il sistema ed implementarlo nel nostro sistema. Purtroppo non siamo riusciti a fare ciò a causa di problemi con il linguaggio di programmazione e con l'implementazione nel sistema.

Teoricamente ciò che ci eravamo preposti fare era questo:

Nel momento in cui un utente visualizzava la scheda di un determinato prodotto noi avremmo ricavato il cluster al quale esso apparteneva ed avremmo consigliato due prodotti appartenenti a quello stesso cluster (quindi molto in relazione con il prodotto che l'utente stava visualizzando).
